A Priority Queue ADT

Name: PriorityQueue <G>

Set: PQ: set of Priority queues containing items from G

G: set of items that can be in the queue

B: {true, false}

N: set of non-negative integers

Signatures:

newPriorityQueue<G>(n): $\longrightarrow$ PQ

PQ.insert(g): G $\nrightarrow$ PQ

PQ.isEmpty: $\longrightarrow$ B

PQ.isFull: $\longrightarrow$ B

PQ.maxItem: $\nrightarrow$ G

PQ.minItem: $\nrightarrow$ G

PQ.deleteMax: $\nrightarrow$ PQ

PQ.deleteAllMax: $\nrightarrow$ PQ

PQ.deleteMin: $\nrightarrow$ PQ

PQ.frequency: G $\nrightarrow$ N

Preconditions:

For all pq ∈ PQ , g ∈ G , n ∈ N

newPriorityQueue<G>(n): none

pq.insert: pq is not full

pq.isEmpty: none

pq.isFull: none

pq.maxItem: pq is not empty

pq.mimItem: pq is not empty

pq.deleteMax: pq is not empty

pq.deleteAllMax: pq is not empty

pq.deleteMin: pq is not empty

pq.frequency: pq is not empty

Semantics:

For pq ∈ PQ , g ∈ G , n ∈ N

newPriorityQueue<G>(n): make a new queue of item from G with capacity n

pq.insert(g): inserts an element g with a certain priority

pq.isEmpty: return true if pq is empty, false otherwise

pq.isFull: return true if pq is full, false otherwise

pq.maxItem: obtain the item in the pq with the highest priority

pq.minItem: obtain the item in the pq with the lowest priority

pq.deleteMax: remove from the pq the item with the highest priority

pq.deleteAllMax: remove from the pq all items that are tied for the highest priority

pq.deleteMin: remove from the pq the item with the lowest priority

pq.frequency: obtain the number of times a certain item occurs in the pq (with any priority)

Question2

a) Algoirthm insert(H, e)

Cost of A.O: O(1)

Let's assume it is a heap with total number of n elements, and k level of height, therefore In this heap, the worst case will be searching from top to bottom, which is k time.

Then the total number of elements n = 2^0 +2^1 +2^2 +……+2^(k-1)

n = 2^k – 1 (sum of 2^0 to 2^(n-1) is (2^n)-1)

k = log(n+1)

So number of execution of loop: log(n+1)

Totalnumber of execution: 1+1+log(n+1) = 2+log(n+1)

O(2+log(n+1)) = O(max(2), (logn+1)) = O(log(n+1)) = O(log(n))

b) Algorithm deleteItem(H)

Since a) & b) are both heap methods, the answer of b) will be the same as a), which is O(log(n)).